Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Especialidad en Sistemas Embebidos

Sistemas Embebidos en el IoT

TRABAJO RECEPCIONAL que para obtener el **DIPLOMA** de
ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: **LUIS JORGE VALENCIA LUNA**

Asesor **DR. JORGE ARTURO PARDIÑAS MIR**

Tlaquepaque, Jalisco. mayo de 2021

Sistemas Embebidos en IoT

Valencia Luna Luis Jorge
Departamento de Electrónica, Sistemas e Informática
Especialidad en Sistemas Embebidos
Tlaquepaque, México
se729982@iteso.mx

***Abstract*— En este documento hablaremos sobre 2 temas que tuvieron su origen separado uno de otro, pero en la actualidad es muy difícil concebir uno sin incluir al otro; Sistemas embebidos e Internet de las cosas.**

Los sistemas embebidos son sistemas que están encargados de cumplir una tarea en específico, generalmente estos sistemas cuentan con recursos limitados y en sus inicios estos dispositivos contaban con limitadas opciones de comunicación con el exterior e incluso con otros dispositivos. Con los avances tecnológicos en la industria estos dispositivos ya son capaces de comunicarse con el usuario e incluso tener comunicación maquina a máquina a través del internet.

Al incluir esta comunicación con el exterior a través del internet hizo que estos dispositivos formaran parte del internet de las cosas, es decir, dispositivos capaces de comunicarse con otros dispositivos y compartir información entre todos los dispositivos en la red.

Hoy en día hablar de Internet de las cosas sin incluir los sistemas embebidos es algo imposible, los sistemas embebidos ya forman parte del internet de las cosas y pueden generar redes incluso sin la necesidad de que el humano interactúe con ellos.

Keywords*— *IoT, Sistemas embebidos, protocolos de comunicación.

I. Introducción

Los sistemas embebidos son sistemas electrónicos digitales basados en microcontroladores que están encargados de realizar una tarea en específico, comúnmente esta tarea no suele cambiar con el paso del tiempo por lo que los sistemas embebidos raramente son actualizados durante su vida útil, al menos este era el concepto inicial de los sistemas embebidos. Esto, aunque en muchas ocasiones forma parte del concepto de los sistemas embebidos ha ido cambiando con el paso de los años y con los avances de la industria.

En los inicios de los sistemas embebidos los microcontroladores contaban con recursos muy limitados en cuestión de comunicación y en la mayoría de los casos la comunicación era meramente utilizada para interactuar con el entorno y no para actualizar su software. Con el paso de los años ha sido posible incluir en los sistemas embebidos comunicación inalámbrica y, con esto, conexión a internet, por lo tanto, los sistemas embebidos ahora forman parte de un sistema aun mayor como es el Internet de las cosas.

El internet de las cosas es un concepto relativamente nuevo pero que va creciendo de forma exponencial ya que el IoT (por sus siglas en inglés) se refiere a todos los dispositivos electrónicos que están conectados al internet y son capaces de intercambiar/comunicarse entre ellos.

Al inicio, el Internet de las cosas no era concebido como una red donde los dispositivos se comunicarán entre ellos sin necesidad de un humano de por medio ya que, aunque la comunicación era comunicación maquina a máquina, había un humano de por medio notificando o instruyendo al dispositivo sobre qué hacer.

Hoy en día eso ha cambiado con la incorporación de los sistemas embebidos al internet de las cosas, si bien muchas veces hay un usuario utilizando el dispositivo conectado al internet, la mayoría del tiempo la comunicación entre dispositivos en el internet de la cosa se ejecuta sin la necesidad de un usuario. Estos dispositivos son capaces de compartir e intercambiar información sin la necesidad de un humano.

Debido a esto, los protocolos han sufrido una revolución para el internet de las cosas y más en los sistemas embebidos. Uno de los mayores retos del internet de las cosas es compartir la información de forma segura entre dispositivos. Los protocolos de los sistemas embebidos en un inicio, debido a sus limitantes en recursos y su naturaleza, no estaban creados para ser protocolos que estuvieran en el internet donde existen muchos usuarios o dispositivos capaces de ver el mensaje, sino que la seguridad del protocolo en el sistema embebido estaba más pensada en la seguridad del mismo hardware. Debido a esto, se han creado protocolos de comunicación pensados en el Internet de las cosas y a su vez ligeros para que puedan ser portados a los sistemas embebidos.

En este documento queremos profundizar en los sistemas embebidos y su unión con el internet de las cosas y como los protocolos juegan un papel importante dentro del sistema embebido, en especial trataremos 2 protocolos en especial: MQTT y Thread. Estos protocolos se ha demostrado que son aptos para los sistemas embebidos ya que no consumen demasiados recursos y son sencillos de acoplar en los sistemas embebidos.

II. Sistemas Embebidos

A. *¿Qué son?*

Cuando la gente escucha el término computadora lo primero que pensamos es en laptops o PC's, incluso recientemente la gente pude llegar a pensar en sus teléfonos inteligentes o tablets

con el término computadora; Pero es muy raro que la gente piense en todos los dispositivos electrónicos a su alrededor con el termino computadora. Sin embargo, la mayoría de los aparatos electrónicos con los que interactuamos hoy en día son controlados por una computadora [3].

¿Entonces, cuál es la diferencia entre una computadora como la laptop y una computadora en los dispositivos electrónicos que usamos día a día? Para esto debemos clasificar las computadoras en 2 categorías, las computadoras de uso general y los sistemas embebidos [3].

Vamos a iniciar con las computadoras de uso general. Estas computadoras están diseñadas para ejecutar cualquier tipo de programa que el usuario desea. El usuario puede instalar, desinstalar y actualizar cualquier software de acuerdo con sus necesidades. Para soportar la gran cantidad de programas que la computadora puede correr este tipo de computadoras cuentan con gran cantidad de recursos, entre ellos, sistemas operativos [3].

Ahora, pensemos en los sistemas embebidos. Los sistemas embebidos son sistemas que no necesitan correr iOS o Windows ya que son sistemas operativos creados específicamente para computadoras de uso general, los sistemas embebidos son sistemas que necesitan responder a impulsos que vienen del mundo exterior y la habilidad de controlar salidas o subsistemas para cumplir con una tarea. En estos casos, la computadora cuenta con los recursos suficientes para cumplir la tarea. Este tipo de computadoras se llaman "computadoras embebidas" también conocidas como microcontroladores o MCU's [3].

Teniendo esto en cuenta podemos decir que un sistema embebido es un circuito electrónico computarizado que está diseñado para cumplir una labor especifica en un producto [1]. Para diferenciar entre una computadora de uso general o un sistema embebido hay ciertas características que debe tener un microcontrolador, por ejemplo, su tamaño, recursos, arquitectura, etc.

Vamos a enlistar las características más generales que debe de cumplir un microcontrolador en la siguiente sección.

B. Características

Una de las principales características de un microcontrolador es que, debido a que no necesita gran cantidad de recursos, como memoria, puede ser implementado todo en el mismo chip [3].

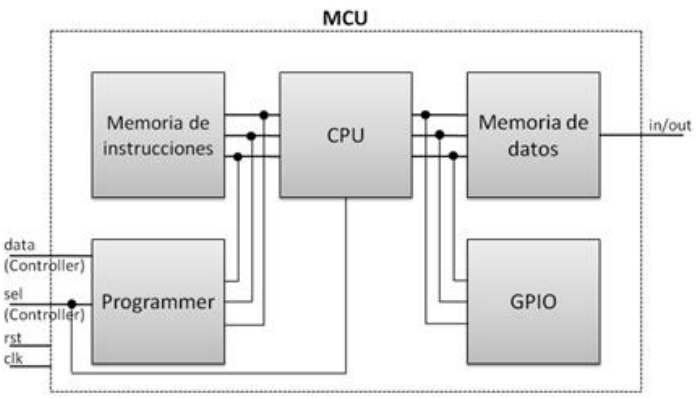

Figura 1.- Arquitectura interna de un Microcontrolador [6].

Además, los sistemas embebidos incluyen gran cantidad de periféricos como timers, convertidores analógicos a digital, convertidores digitales a análogos, interfaces para comunicación serial, etc, lo cual los hace muy versátiles para controlar todo tipo de dispositivos electrónicos [3].

Otra característica de los sistemas embebidos es que no están diseñados para correr software de forma arbitraria a deseo del usuario, sino que el SW que corre en los sistemas embebidos está diseñado para el sistema en sí. El software que corre en los sistemas embebidos es llamado "Firmware" para recalcar que el software no requiere ser cambiado frecuentemente. Incluso ciertos sistemas embebidos, que son diseñados para poder actualizar su software, no es algo que comúnmente sucede, es decir hay una estrecha relación entre el software del sistema embebido y el hardware en el que se ejecuta [3].

Si bien los sistemas embebidos pueden tener sistemas operativos, no son nada similar a un IOS o un Windows como los que tienen las computadoras de uso general. En su lugar, los sistemas operativos de los sistemas embebidos actúan más como un calendarizador de tareas encargado de coordinar actividades de tiempo real, como lectura de botones o sensores. Los sistemas operativos de los sistemas embebidos son llamados "Sistemas operativos de tiempo real" [3].

C. Middleware y su importancia en los Sistemas Embebidos

Si bien ya hemos hablado de las características de los sistemas embebidos, hay una característica muy propia que juega un papel fundamental en todos los sistemas embebidos: La arquitectura de SW.

Existen muchos tipos de arquitecturas en los sistemas embebidos y usualmente la arquitectura a utilizar en el sistema está muy ligada al acoplamiento entre el Hardware y el Software. Sin embargo, la gran diversidad en las arquitecturas de Software causa distintos problemas como son la portabilidad, el reúso, mantenimiento, dependencia, etc [7].

Más aún, la comunicación y problemas de intercambio entre sistema operativo y librerías e interfaces entre librerías a sistema operativo son retos significativos [7].

Las arquitecturas tradicionales en los sistemas embebidos están compuestas por seis módulos que se muestran en la figura 2. Estos módulos, iniciando desde la capa más baja serían: Hardware, capa de periféricos, sistema operativo, archivos de

sistema, librerías y aplicación. La utilización de las arquitecturas tradicionales presenta varios problemas a la misma arquitectura: El reúso de componentes, el intercambio de información entre capas, la portabilidad de los módulos de software, la dependencia entre plataformas y la flexibilidad entre los sistemas [7].

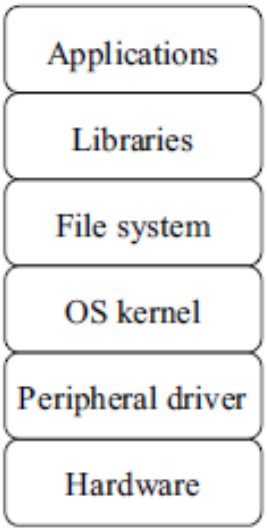

Figura 2.- Arquitectura tradicional de sistemas embebidos [7].

Para resolver este tipo de problemas, a las arquitecturas de sistemas embebidos se les agrega una capa comúnmente llamada "middleware". Esta capa lo que propone es proveer interfaces para poder comunicar la capa de aplicación con la capa de servicios de sistema, unificando las interfaces para las arquitecturas y con eso resolver los problemas de portabilidad, reúso y dependencias de hardware.

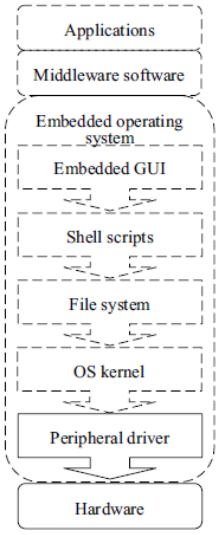

Figura 3.- Arquitectura utilizando middleware [7].

Esta arquitectura propone una separación en tres principales capas: Aplicación, Middleware y servicios de sistema como se muestra en la figura 3.

Los protocolos de comunicación en los sistemas embebidos son configurados y preparados en la capa de los periféricos, es decir que en esta capa se inicializan los transceivers, se configuran los puertos y se inicializa el driver para que el hardware sea capaz de transmitir los datos de comunicación. Una vez inicializado el hardware para la comunicación, se prepara el stack de comunicación. Este stack es el proveedor de interfaces y servicios para que las capas de más alto nivel, comúnmente desarrollados en el middleware, sean capaces de establecer comunicación entre la capa de aplicación y el hardware. Con esto se logra una separación entre la aplicación y el hardware hablando de la comunicación. Si bien el hardware, puede ser generado o configurado para utilizar SPI, UART, I2C o cualquier otro tipo de protocolo de comunicación, para la aplicación esto no es relevante, ya que el middleware lo abstrae de esa configuración.

III. Internet de las cosas

A. *¿Qué es?*

El internet ha ayudado a las personas a conectar con información estática, pero hoy en día está ayudando a generar conexiones entre personas, personas a dispositivos físicos y conexiones entre dispositivos físicos [2].

Para el año 2020 existían más de 30 billones de dispositivos conectados al internet. Sin embargo, 99% de dispositivos físicos siguen sin conexión a internet. El crecimiento exponencial y la convergencia de datos en internet y sus procesos están provocando que las conexiones sean cada vez más relevantes y valiosas [2].

El internet de las cosas es un concepto computacional que describe un escenario donde todos los días objetos físicos están conectados a internet y son capaces de identificarse a sí mismos entre todos los dispositivos conectados en la red [2].

Internet de las cosas es una red de dispositivos capaces de comunicarse entre ellos utilizando dirección IP sin interferencia humana. El internet de las cosas es un ecosistema de objetos inteligentes, dispositivos inteligentes, tabletas, teléfonos inteligentes, etc. [2].

Figura 4.- Grafica de IoT [15]

La idea es intercambiar datos utilizando protocolos especializados para el IoT en sistemas embebidos. La necesidad de protocolos especializados se debe a que los sistemas embebidos tienen capacidades y recursos limitados. El protocolo

debe de ser confiable, bajo consumo de energía y baja cantidad de datos en la red. La misma red es una limitante en la cantidad de datos que pueden viajar en la red. Existen diferentes tipos de comunicación física como son bluetooth, zigbee, WiFi, ethernet, etc. Pero hay otros tipos de protocolos de comunicación con una idea en particular, como son MQTT, TCP, sockets, XMPP, CoAP, Thread entre otros [9]. Estos protocolos en particular son protocolos pensados para dispositivos en el IoT.

B. Características de dispositivos en IoT

Algunas de las características de los dispositivos en el Internet de las cosas las podemos enlistar en [2]:

- Interconectados: La conexión entre usuarios a dispositivo o dispositivo a dispositivo debe ser sencilla
- Censado inteligente: La tecnología de censado debe de dar una verdadera conciencia del mundo físico, personas y objetos.
- Inteligencia: Los dispositivos en IoT deben de tener inteligencia en ellos mismos y entre los dispositivos conectados.
- Bajo consumo de energía: Deben de ser capaces de ahorrar energía y eficientizar su consumo por medio del manejo de su energía total.
- Comunicación: Capaces de comunicar su estado a los otros dispositivos conectados a su alrededor.
- Seguridad: Deben de ser capaces de garantizar la seguridad del usuario.

C. Retos de IoT en los sistemas embebidos

Si bien el internet ha venido a revolucionar la industria de los dispositivos electrónicos, también viene acompañada de varios retos que han ido incrementando con el crecimiento de los sistemas conectados a internet. Cuando hablamos de sistemas embebidos en el internet de las cosas, estos retos son mayores por las limitantes que ya conocemos de los sistemas embebidos en sí, es decir sus recursos limitados, el bajo consumo de energía, el tamaño, etc.

Uno de los mayores retos de los sistemas embebidos en el internet de las cosas es el protocolo de comunicación a utilizar. Los dispositivos conectados en el internet de las cosas, que se comunican entre ellos, deben de aceptar diferentes aspectos para tener un intercambio efectivo de mensajes bajo ciertos protocolos [8].

Con el paso de los tiempos se han ido desarrollando cada vez más protocolos para el IoT debido a que los anteriores eran inadecuados para cumplir los requisitos del IoT. Hay ciertos criterios que los protocolos deben de contemplar para que sean una opción dentro del IoT: escalabilidad, comunicación, manejo, seguridad, etc, son algunos de los parámetros a considerar en un protocolo a usar en el IoT [8].

Para resolver este tipo de retos el IoT ha desarrollado la mayoría de sus protocolos de comunicación en la capa de aplicación. Esta estrategia hace que los protocolos de comunicación sean más flexibles para el IoT y que a su vez no dependan de la capa física del sistema embebido, agregando portabilidad y reduciendo dificultades de manejo.

La ventaja de utilizar este tipo de enfoque es que los protocolos en la capa de aplicación proveen servicios que son muy efectivos en la comunicación para dispositivos con bajo consumo de energía y de bajo costo. Esto también genera un set de protocolos los cuales son capaces de intercambiar mensajes a nivel aplicación entre diferentes dispositivos [8].

IV. Protocolos en el IoT

Puede decirse que las comunicaciones son el centro del género IoT. Existe una relación en los dispositivos IoT, mientras más complejos y mayor velocidad necesiten los protocolos de comunicación mayor será la cantidad de procesamiento y el consumo de energía del dispositivo [4].

Antes de hablar de la comunicación en el IoT es necesario entender el modelo de comunicación OSI para poder entender cómo se desarrollaron los protocolos de comunicación para el IoT y sobre que capa se desarrollan.

El Modelo OSI puede entenderse como un modelo conceptual de comunicación entre dispositivos que se encuentran conectados entre sí formando una red de comunicaciones y fue creado por la Organización Internacional de Estandarización para permitir que diversos sistemas de comunicación se comuniquen entre sí usando protocolos estándar. El modelo OSI propone 7 capas para dividir el sistema de comunicación, las capas son:

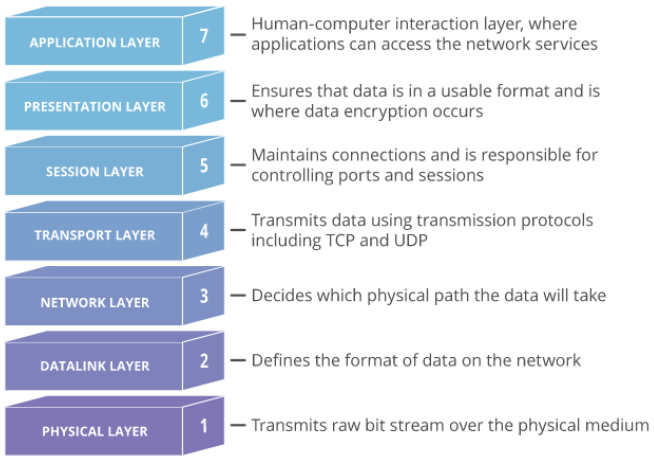

Figura 5.- Capas de comunicación del modelo OSI [13].

Dentro de las capas del Modelo OSI podemos observar que mientras más baja la capa está más ligada al hardware, es decir a una tecnología en específico y la capa más alta es la más separada del hardware y, por lo tanto, no depende de una tecnología o no está ligada a al hardware.

Mucho antes de que el IoT apareciera ya existían comunicaciones máquina a máquina (M2M "Machine to Machine" en inglés) la cual la transferencia de datos no depende de que una persona interactúe directamente en ella. Los protocolos de IoT tienen diferentes capacidades y diferentes funciones, lo que los hace capaces de cumplir con diferentes aplicaciones [5].

La mayoría de los protocolos utilizados en el IoT son protocolos situados en la capa de Aplicación a nivel OSI, esto hace que el protocolo sea más flexible y no dependa de una capa física para funcionar. Al ser protocolos de comunicación en la capa de aplicación pueden ser protocolos que su empaquetado/desempaquetado y las adaptaciones necesarias puedan hacerse en la capa más alta del modelo OSI y sus telegramas sean adaptados a la necesidad de la aplicación, algunos de los protocolos de comunicación más relevantes son:

- **MQTT (Message Queueing Telemetry Transport)**: Protocolo basado en Publicador/Suscriptor diseñado para comunicación maquina a máquina de baja carga de datos. La arquitectura de MQTT es Cliente/Servidor, donde cada sensor es un cliente conectado a un servidor. El servidor es conocido como "Broker" y utiliza comunicación TCP. MQTT es un protocolo con mensajes orientados y cada mensaje es publicado en tema en específico. Los clientes se pueden suscribir a varios temas y cada cliente suscrito a un tema recibe cada mensaje que se publica a ese tema en específico [5].

- **CoAP (Constrained Application Protocol):** CoAP fue diseñado como un protocolo para dispositivos con limitantes, es similar a HTTP, pero a diferencia de HTTP CoAP usa paquetes de información más pequeños. Este protocolo utiliza mucho las conversiones de string a enteros y los campos a nivel bit para guardar espacio en sus envíos de mensajes. Los mensajes son muy sencillos de generar y no consumen RAM adicional. CoAP es un protocolo que trabaja con UDP. Los clientes y servidores trabajan con datagramas sin seguridad y la entrada y reordenamiento de información es hecha a nivel aplicación. CoAP está basado en el funcionamiento Cliente/Servidor donde el cliente puede utilizar los comandos GET, PUT, POST y DELETE [5].

- **XMPP (Extensible Messaging and Presence Protocol):** Protocolo basado en XML. Es un protocolo en tiempo real el cual genera una amplia gama de aplicaciones, como son mensajería, presencia, chats privados, colaboración, middleware ligero, contenido y generalidades de XML [5].

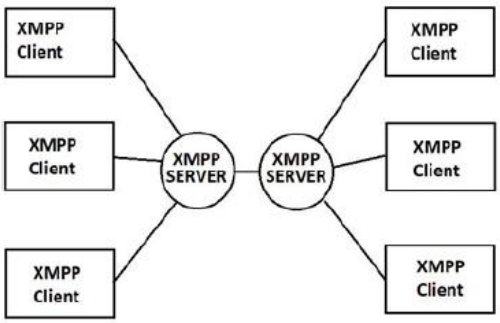

Figura 6.- Ejemplo de protocolo XMPP [5].

- **HTTP (Hypertext Transfer Protocol):** HTTP es la base de cliente/servidor utilizado en la WEB. HTTP puede ser muy seguro si generas dispositivos IoT solo como clientes y no como servidores, es decir, si se utiliza HTTP para iniciar conexiones y no para recibir conexiones. Se podría decir que es muy seguro cuando se usa para una red LOCAL. HTTP utiliza el protocolo TCP y garantiza fiabilidad de entrega. HTTP no es un protocolo utilizable para dispositivos con pocos recursos debido a su alto consumo de energía y tiempo de ejecución. HTTP fue diseñado como un protocolo para la comunicación entre 2 dispositivos a la vez. En la mayoría de las aplicaciones de IoT existen gran cantidad de sensores conectados al mismo tiempo generando información que debe ser consumida por algún otro sensor, HTTP no cumple con este requisito [5].

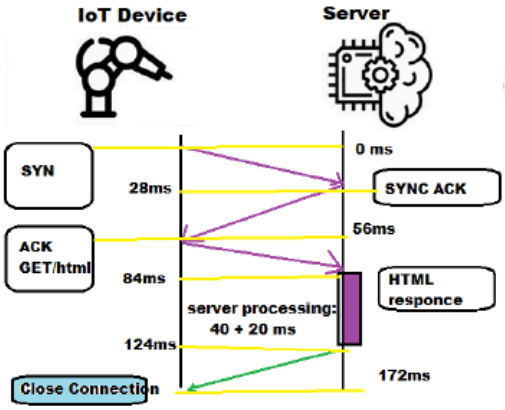

Figura 7.- Protocolo HTTP abriendo y cerrando conexión entre servidor y dispositivo IoT [5].

- **Thread**: Thread Group, Inc. Fue fundada por siete compañías, ARM (Softbank), Big ass Fans, Freescale (NXP), Nest Labs (Google), Samsumg, SiliconLabs y Yale locks. La especificación de Thread v1.0 fue publicada en el 2015 por Thread Group, Inc. Actualmente Thread Group, Inc. tiene más de 200 miembros en todo el mundo [11]. Thread está pensado como un estándar abierto y está construido incorporando varios de los estándares actuales y de los estándares propuestos. La especificación completa de Thread se encuentra disponible en Thread Group, Inc. [11]. Thread es un protocolo que está basado en dispositivos de bajo consumo de energía, lo cual lo hace una gran opción para los sistemas embebidos en el IoT además de ser amigable con el usuario y sencillo para agregar y remover nodos de la red.

- **Zigbee**: Zigbee es el nombre de un conjunto de protocolos de comunicación de alto nivel inalámbrica con radiodifusión digital de bajo consumo, basado en IEE 802.15.4 de redes inalámbricas de área personal. Su objetivo son las aplicaciones que requieren comunicación segura y

de bajo consumo de datos para maximizar la vida útil de las baterías [12].

- **Bluetooth:** Bluetooth es una especificación industrial para redes inalámbricas de área personal (WPAN) creada por Bluetooth Special Interest Group, Inc. que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace de radiofrecuencia en la banda ISM de 2.4 Ghz [12].

V. THREAD Y MQTT EN EL IOT

Debido a las facilidades que ofrecen MQTT y Thread estos fueron los protocolos utilizados en la especialidad para probar la comunicación de los dispositivos en el IoT.

Para el desarrollo de este tema vamos a ahondar en estos 2 protocolos de comunicación.

A. THREAD

Thread es un protocolo de comunicación basado en protocolos de red IP. Este protocolo es construido sobre tecnologías que ya han sido probadas: 6LoWPAN, UDP y sobre IEEE 802.15.4. Thread ofrece la facilidad de agregar y remover dispositivos a la red creada y puede ser escalable hasta más de 250 dispositivos, además de que Thread es utilizado sobre dispositivos de muy bajo consumo de energía.

El protocolo Thread es regulado por un grupo sin fines de lucro el cual es el encargado del mercado de educación y de certificación de productos que usen el protocolo Thread.

Figura 8.- Thread group Inc [14].

Para que puedas ser certificado como un producto Thread tienes que cumplir con los siguientes requisitos:

- **Bajo consumo de energía**
- **Resilencia**
  - **Sin fallas de un solo punto**
  - **Posibilidad de curarse a sí mismo**
  - **Robusto ante interferencias**
  - **Extensible por sí mismo**
  - **Confiable en cualquier infraestructura**
- **Direccionamiento por IP**
- **Protocolo abierto**
- **Seguro y amigable**
- **Disponible al mercado**
- **Usar tecnologías existentes**

La arquitectura de red de Thread está basada en la arquitectura de red en malla:

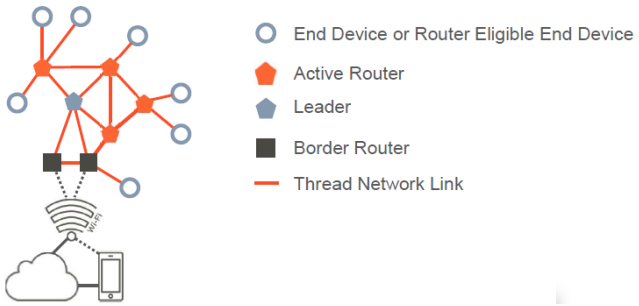

Figura 9.- Arquitectura de red Thread [14].

Dentro de la arquitectura de red Thread, un dispositivo puede tomar diferentes roles e incluso puede haber hasta 16,000 dispositivos Thread en una red:

- **End Device o RouterElegible End Device:** Diseñado para bajo consumo, puede estar encendido o apagado, puede convertirse en "Router elegible" si se encuentra encendido. Pueden haber más de 512 dispositivos por cada router activo.
- **Active router:** Encargados de enrutar el tráfico de datos entre los dispositivos Thread. Son la columna vertebral para la red de malla, estos dispositivos también son elegibles como líder. Puede haber hasta 31 en la red.
- **Leader:** Maestro de parámetros de red, Coordina a los comisionados y enruta el tráfico entre dispositivos. Un dispositivo en la red.
- **Border router:** Los enrutadores de frontera son los encargados de mandar los datos desde la nube y también pueden proveer conectividad Wi-fi.

Thread es un protocolo muy ágil, esto significa que un dispositivo puede cambiar de rol en tiempo de ejecución lo cual lo hace un protocolo muy dinámico y amigable ya que esta agilidad se puede ir adaptando a las necesidades del usuario.

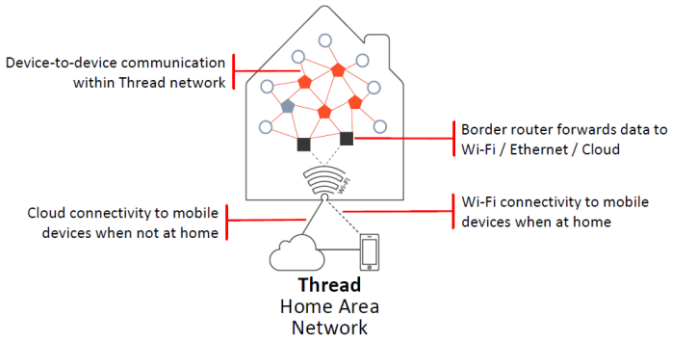

Figura 10.- Modelo de comunicación de una red Thread [14].

Actualmente Thread esta siendo el protocolo que rompe los paradigmas de los protocolos en el internet de las cosas, como ya se ha mencionado, los mayores retos en el IoT es que no existe una estandarización de protocolos en el IoT. Thread es

un protocolo que resuelve los problemas de los protocolos de comunicación en el IoT al ser:

- Protocolo de estándar abierto
- Sencillo de utilizar
- Seguro y encriptado
- Eficiente consumo de energía
- Sin fallas de un solo punto
- Diseñado para soportar gran variedad de productos.

Se ha mencionado que la mayoría de los protocolos para IoT está desarrollado en la capa de aplicación, en el caso particular de Thread, este abarca más capas a nivel OSI, que va desde el nivel físico hasta la capa de transporte, esto hace de Thread un stack muy completo y sobre el cual se pueden montar otros protocolos a nivel aplicación, por ejemplo, MQTT.

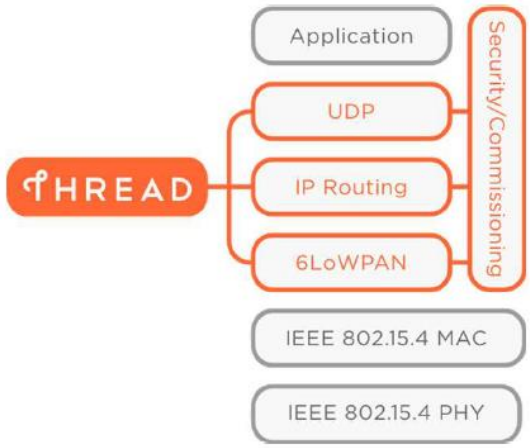

Figura 11.- Stack de protocolo Thread [1].

Thread no especifica el uso especial de un protocolo a nivel de aplicación, por lo que Thread se vuelve una buena opción a utilizar en IoT cuando el protocolo a nivel aplicación todavía no esta definido.

B. MQTT

El protocolo MQTT es un protocolo ligero y flexible que puede ser implementado en dispositivos pequeños. MQTT requiere bajo consumo de energía y de código para su implementación en relación con otros protocolos.

MQTT es un protocolo de publicación/suscripción que es liviano y requiere mínimo ancho de banda para conectar los dipositvos de IoT. A diferencia del paradigma de solicitud/respuesta que conlleva HTTP, MQTT se basa en eventos que permite enviar mensajes a los clientes. Este tipo de arquitectura desacopla a los clientes entre si para permitir una solución altamente escalable.

Los componentes de MQTT se pueden enlistar en:

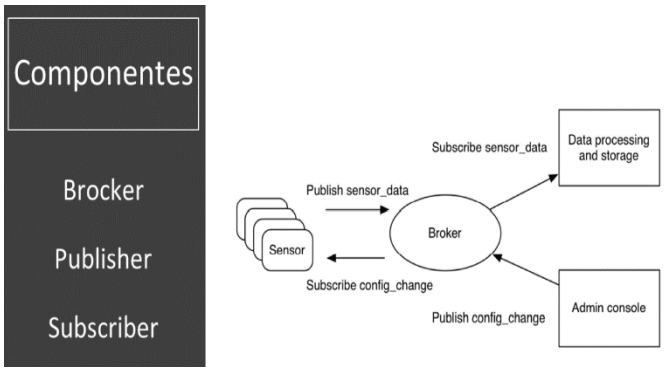

Figura 12.- Componentes en una red MQTT [14].

- **Publisher:** Es el que genera la información
- **Brocker**: Encargados de mandar los mensajes entre remitentes y destinatarios.
- **Subscriber:** Es el que se inscribe a los temas de interés para consumir la información.

MQTT maneja tres tipos de QoS (Quality of service en Ingles) para compensar las redes poco fiables, estos tres niveles se dividen en:

- **QoS 0 Unica vez:** Este nivel de calidad en el servicio solo envía el dtagrama y no espera un acuse de recibido (Acknowledge) por parte de los demás dispositivos en la red.
- **QoS 1 Al menos uno:** En este nivel de calidad de servicio se envía el datagrama hasta que recibe un acuse de recibo (Acknowledge) por parte del orto dispositivo en la red.
- **Qos 2 Exactamente uno:** Este nivel de calidad de servicio envia el datagrama y espera exactamente un acuse de recibo (Acknowledge) por parte del otro dispositivo en la red para continuar con la comunicación.

Figura 13.- Niveles de QoS en MQTT [14].

Una de las ventajas del protocolo MQTT es que incluye "Tópicos" dentro de la información para los suscriptores, estos temas sirven para clasificar la información por temas de interés entre el usuario y el brocker.

VI. THREAD Y MQTT EN LA PRÁCTICA

A. *Thread en la práctica*

Para la práctica de Thread se utilizó la tarjeta NXP K64 sobre la cual se montó la tarjeta CR20A para utilizar la comunicación basada en la IEE 802.15.4 para comunicaciones inalámbricas.

Dentro de la práctica para Thread se utilizaron 3 tarjetas independientes para intercambiar mensajes entre ellas. Las tarjetas serian recursos independientes: "recurso1", "recurso2" y un "líder", sin embargo, las tarjetas no tenían su nombre estático, sino que si una tarjeta intentaba conectarse a la red y no había nadie en ella se convertía en líder y las tarjetas que se unieran a la red iniciaban como "recurso1" o "recurso2" si ya había una tarjeta conectada a la red.

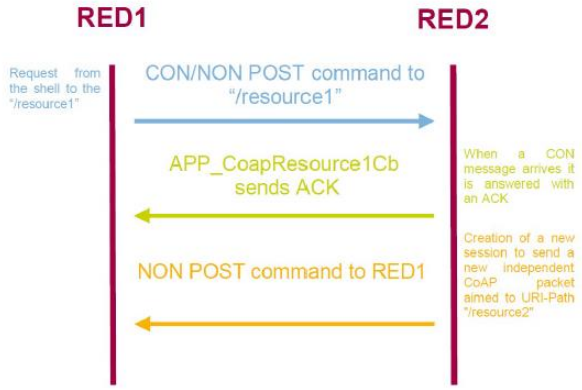

Figura 14.- Comportamiento esperado tarjetas Thread [14].

Dentro de la práctica las tarjetas utilizarían el método "CON" para enviar mensajes, a lo que se debería contestar con un acuse de recibo "ACK". Todos los mensajes que se enviaran al recurso 1 enviaban su réplica al recurso 2 con un método "NON POST", es decir sin acuse de recibo, a su vez se debía de imprimir los mensajes enviados recibidos en la consola.

El primer paso dentro de la práctica de Thread fue generar los recursos de nuestra tarjeta desde el punto de vista Thread, esto es, los temas a los que nuestra tarjeta va a responder en dado caso que algún otro dispositivo lo requiera:

```
#define APP_RESOURCE1_URI_PATH          "/resource1"    //LuVa
#define APP_RESOURCE2_URI_PATH          "/resource2"    //LuVa
#define APP_OPEN_DOOR_URI_PATH          "/Open_Door"    //LuVa
#define APP_VIOLATION_URI_PATH          "/Violation"    //LuVa
#define APP_KEY_PASSWORD_URI_PATH       "/KeyPassword"  //LuVa
#define APP_LIGHT_URI_PATH              "/Light"        //LuVa
```

Figura 15.- Recursos dentro de la aplicación Thread.

Una vez teniendo los recursos o temas de nuestro dispositivo se tienen que generar los callbacks de estos recursos, estos callbacks son los encargados de la funcionalidad del recurso en sí:

```
static void APP_CoapResource1Cb(coapSessionStatus_t sessionStatus, void *pData, c
static void APP_CoapResource2Cb(coapSessionStatus_t sessionStatus, void *pData, c
static void APP_Coap_Open_DoorCb(coapSessionStatus_t sessionStatus, void *pData,
static void APP_Coap_ViolationCb(coapSessionStatus_t sessionStatus, void *pData,
static void APP_Coap_GetPasswordCb(coapSessionStatus_t sessionStatus, void *pData
static void APP_Coap_GetTemp(coapSessionStatus_t sessionStatus, void *pData, coap
```

Figura 16.- Callbacks de los recursos del dispositivo.

Después de estos 2 pasos, se genera la relación entre el callback y el recurso que se generó, esta conexión será la encargada de llamar el callback cuando se reciba de otro dispositivo por el recurso de interés:

```
{APP_CoapSinkCb, (coapUriPath_t *)&gAPP_SINK_URI_PATH},
{APP_CoapResource1Cb, (coapUriPath_t*)&gAPP_RESOURCE1_URI_PATH}, //Luva
{APP_CoapResource2Cb, (coapUriPath_t*)&gAPP_RESOURCE2_URI_PATH}, //LuVa
{APP_Coap_Open_DoorCb,(coapUriPath_t*)&gAPP_OPEN_DOOR_URI_PATH}, //Luva
{APP_Coap_ViolationCb,(coapUriPath_t*)&gAPP_VIOLATION_URI_PATH}}; //LuVa
```

Figura 17.- Relación entre callback y recurso.

Una vez hecho esto ya existe la relación entre los recursos y los callbacks para la ejecución de la funcionalidad deseada. Las llamadas a Thread son parte ya de un stack que se entrega por parte del proveedor, al ser Thread un protocolo establecido y con cierto interés de que sea incorporado a grandes escalas, los proveedores de microcontroladores, que incorporan Thread, proveen un stack de Thread para que sea más sencillo de utilizar.

Con esto definido, lo único faltante es la implementación de la funcionalidad de los callbacks, lo cual como estudiante e interesado de aprender protocolos de comunicación en IoT y sistemas embebidos Thread es más sencillo de utilizar.

Thread se comportó como un protocolo muy amigable, sencillo de implementar y utilizar. La red se conformó de manera dinámica sin necesidad de un esfuerzo extra al momento de desarrollar.

B. *MQTT en la práctica*

Durante la especialidad de sistemas embebidos utilizamos la tarjeta K64 de NXP para implementar y aprender del protocolo MQTT en el IoT. La práctica consistía en interconectar un teléfono móvil, un brocker en la nube y la tarjeta K64 para intercambiar información entre ellos.

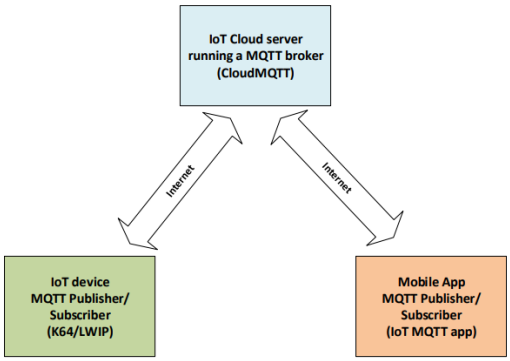

Figura 18.- Practica de MQTT [14]

Como proveedor de servicio en la red se seleccionó MQTTcloud debido a que es un servicio gratuito además de que es amigable para el manejo de tópicos y usuarios, lo que lo hace un sistema muy sencillo de utilizar para los fines educativos. Para la aplicación móvil, debido a que IOS no cuenta con aplicaciones que sean sencillas de utilizar, se optó por utilizar una terminal de MQTT, aunque no era una aplicación que nos diera las facilidades de agregar botones o hacerlo más visual, cumplía con los objetivos que era intercambiar los mensajes entre el Brocker y la tarjeta a través de los tópicos de interés.

Como práctica se simuló un sistema de alarma de un hogar, la tarjeta K64 fungía como el sistema de control de la alarma dentro del Hogar mientras que la aplicación en el teléfono nos permitía controlar la alarma a través del brocker, en este caso MQTTcloud. La comunicación entre aplicación móvil y Alarma (K64) no era directa, sino que todo se hacía a través del brocker.

La prática consistió en conectar una tarjeta al internet utilizando Ethernet a través del protocolo TCPIP que a su vez utilizaba el stack de Thread. Una vez inicializado el sistema se iniciaba la conexión con el brocker y se generaba una pequeña red entre Brocker, K64 y teléfono móvil.

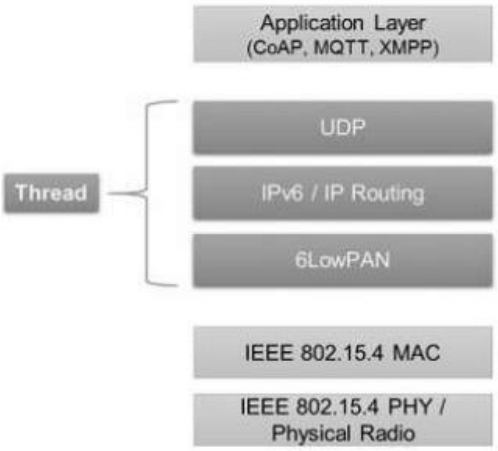

Figura 19.- Implementación de MQTT en la práctica [14].

Si bien MQTT es un protocolo sencillo y creado para sistemas embebidos hay ciertos puntos que al momento de realizar la práctica dieron un poco de problema. Para la parte de la tarjeta K64 se configuró la MAC address, la dirección del Brocker, un usuario y password para la comunicación con el brocker, los tópicos a los que estaba suscrito y los tópicos que la tarjeta publicaba y el puerto sobre el cual iba a enviar y recibir información.

El problema de la configuración que utilizamos fue que se hizo de manera estática en el código por limitantes del hardware e incluso un poco falta de conocimiento o no correcta planeación de la arquitectura para la práctica, para fines de una Alarma de casa no representa un problema ya que es un sistema que no requiere muchas actualizaciones durante su vida útil, pero dentro del internet de las cosas y con los avances tecnológicos una configuración dinámica de tópicos, servidor y puerto da más facilidades al usuario y al sistema en sí.

La práctica fue muy interesante y retadora debido a que MQTT es prácticamente un protocolo nuevo para los sistemas embebidos, además de que nos mostró que actualmente la distancia no es una limitante para los sistemas embebidos como originalmente lo era debido a limitantes del hardware.

VII. Conclusiones

MQTT y Thread son protocolos creados especialmente para IoT y los 2 protocolos son sencillos de integrar en los sistemas embebidos. Pero hay ciertas particularidades entre los 2 protocolos las cuales se deben de tomar en cuenta al momento de desarrollar el dispositivo electrónico.

Iniciaremos hablando con MQTT. MQTT es un protocolo que está diseñado para conectar el dispositivo electrónico a la nube, es decir a un brocker. Esto se tiene que tomar en cuenta ya que muchas veces nuestras aplicaciones no necesitan esa funcionalidad y en este caso MQTT no sería una opción para utilizar.

Una de las ventajas de MQTT es que puede ser utilizado para productos de bajo consumo de energía, además para implementar este stack no son necesarios muchos recursos y puede aplicarse fácilmente a un sistema embebido.

MQTT puede disponer de los mensajes y los temas que se encuentran en el brocker con sus servicios publis/request y suscribe y mientras estén en el brocker la información puede ser obtenida. Sin embargo, hay ciertas desventajas en MQTT, el protocolo no soporta un Time To Live (TTL) en los mensajes y el encolado de los mensajes es muy pobre, de hecho, solo se encola un mensaje, esto quiere decir que si se quiere tener historial de los mensajes enviados o recibidos no es posible salvo por el último mensaje, el cual puede ser obtenido a través del brocker. Así que, si dentro de tu aplicación es necesario obtener información de mensajes encolados, MQTT no es la opción adecuada.

Hablando ahora del protocolo Thread, una de las principales diferencias es que este protocolo no está diseñado para conexión a la nube, sino que es un protocolo para conexión entre dispositivos.

Thread también es un protocolo que se puede utilizar para sistemas de bajo consumo de energía y baja cantidad de recursos, de hecho, una de las características de Thread es que muchas veces los recursos a utilizar dependen más del mismo dispositivo y su funcionalidad y no tanto del protocolo en sí.

Thread fue un protocolo más sencillo y amigable desde el punto de vista de desarrollador, Una vez integrado y desarrollado tu dispositivo las conexiones entre dispositivos se hacen de manera dinámica y no es algo que se deba de considerar de manera estática al momento de desarrollar.

Los 2 protocolos cumplen funcionalidades diferentes, uno es para conexión con la nube y otro para la conexión entre dispositivos, sin embargo, desde el punto de vista experimental y educacional, Thread parece un protocolo más adecuado para adentrarse en el mundo de los sistemas embebidos y el IoT ya que no se depende de un brocker y se puede probar la arquitectura completa de Thread teniendo diferentes dispositivos en la misma red.

References

[1] Gustavo Galeano, “Programación de sistemas embebidos en C”, primera edición, Alfaomega grupo editor, México, julio 2009

[2] Sachchidanand Singh, Nirmala Singh, “Internet of Things(IoT): Security Challenges, Business Opportunities & Reference Architecture for E-commerce”, International conference on Green computing and Internet of Things, 2015

[3] B. J. Lameres, “Embedded Systems Design using the MSP430FR2355 LaunchPad”, Springer Nature, Switzerland 2020.

[4] John C. Shovic, “Raspberry Pi IoT Projects”, 2021

[5] Neven Nikolov, “Research of MQTT, CoAP, HTPP and XMPP IoT Communication protocols for Embedded Systems”, Department of computer System and Technology, Bulgaria, September 16 2020.

[6] Mauro CIPOLLONE, Carlos E. MAIDANA, Fernando I. SZKLANNY, “Desarrollo de microcontrolador embebido en FPGA”, Universidad Nacional de la Matanza, diciembre 2016.

[7] Yang-Hsin Fan, Jan-Ou Wu, “Middleware Software for Embedded Systems”, 26th International Conference on Advanced Information Networking and Applications Workshops, 2012.

[8] Cheena Sharma, Dr. Naveen Kumar Gondhi, “Communication Protocol Stack for Constrained IoT Systems”, Shri Mata Vaishno Devi University.

[9] Neven Nikolov, “Research of the communication protocols between the IoT Embedded system and the Cloud structure”, Faculty of Computer Systems and Technologies, Bulgaria, 2018.

[10] Neven Nikolov, “Research of the communication protocols between the IoT Embedded system and the Cloud structure”, Faculty of Computer Systems and Technologies, Bulgaria, 2018.

[11] Ishaq Unwala, Zafar Taqvi, Jiang Lu, “Thread: An IoT Protocol”, University of Houston Clear Lake, 2018.

[12] https://es.wikipedia.org/ , Abril 2021

[13] https://www.cloudflare.com/es-es/learning/ddos/glossary/open-systems-interconnection-model-osi/#:~:text=El%20modelo%20OSI%20puede%20entenderse,ellos%20apilado%20sobre%20el%20prec, Abril 2021

[14] Sergio Nicolás Santana Sánchez, Material de clase, Septiembre, 2019

[15] https://www.muycanal.com/2020/02/26/mercado-iot-crece-en-espana, Mayo 2021